

APPENDIX I
Copyright © 2000-2001 ARC International plc. All rights reserved.

[illegible]

	bl
	b
	beq
5	bgt
	bhi
	stbblink
	jblink
	jr
10	jlr
	movr
	movf0r
	movf0h
	movrh
	movhr
15	cmprh
	cmphr
	cmpr
	cmpi64
	movi64
20	addi32
	subi32
	addabi8
	subabi8
	subneaaa
25	subhhh
	subaaa
	subaab
	subrrr
30	addaab
	addrrr
	addrrrh
	asli8
	aslab1
	aslab2
35	asri8
	asrab1
	asrab2
	lsri8
	lsrab1
40	lsrab2
	andi32
	andfi32
	andaab
	andfab
45	mul0ab
	muli32
	ldabc
	ldr64
	ldwr32
50	ldbr16
	str64
	stbr8
	stwr16
	ldrpc
55	addrpc
	ldfp32
	stfp32
	addfpi32
	ldgp
60	

APPENDIX II

Copyright © 2000-2001 ARC International plc. All rights reserved.

```

#
# Confidential Information
# Limited Distribution to Authorized Persons Only
# Created 2000 and Protected as an Unpublished Work
5 # Under the U.S. Copyright act of 1976.
# Copyright © 2000-2001 ARC CORES LTD
# All Rights Reserved
#

```

```

# SCCS release : %M% %I% %G%

```

```

# Description : Script to analyse an ARC assembler file and
# print frequency of usage stats for various
# ARC instruction formats

```

```

#--

```

```

=====
=====--#

```

```

BEGIN {
}

```

```

{op[$1]++}

```

```

END {
  OFS="\t"
  for (i in op) print i, op[i], int(op[i]*1000/NR)/10
}

```

```

#/(j|jl|b|bl)(ge|gt|le|lt|ne|eq|pl|mi|hi|hs|lo|ls)?\.d/ {
# stored = $0
# sub(/\.d/, "", stored)
35 # getline
# print $0
# print stored
# next
#}

```

```

40 #
#{ print $0 }

```

APPENDIX III

Copyright © 2000-2001 ARC International plc. All rights reserved.

```

#
# Confidential Information
# Limited Distribution to Authorized Persons Only
# Created 2000 and Protected as an Unpublished Work
5 # Under the U.S.Copyright act of 1976.
# Copyright © 2000-2001 ARC CORES LTD
# All Rights Reserved
#

```

```

10 # SCCS release : %M% %I% %G%
#

```

```

# Description : Script to analyse an ARC assembler file and
# print frequency of usage stats for various
# proposed ARC instruction formats
#

```

```

15 #
#
#--
=====
=====--#

```

```

20 BEGIN {
    out = "c"
    #reg = "%r(0|1|2|3|13|14|15|16),"
    reg = "%r(0|1|2|3|13|14|15|16) ([^0-9]|$)"
25 regh = "%(r[0-9]+|sp|fp|gp|blink) ([^0-9]|$)"
    reg01 = "%r(0|1) ([^0-9]|$)"
    reg23 = "%r(2|3) ([^0-9]|$)"
    reg1316 = "%r(13|14|15|16) ([^0-9]|$)"
    pete = 0
30 printf "" >out
}

```

```

function nxt() {
35 print $0 >>out
    next
}
function nextc() {
    print "c" $0 >>out
    next
40 }

```

```

$1 == "bl" {
    bl++
45 if ($2 ~ /__prolog_.*/) {
        push++
        nxt()
    } else {
        calls[$2]++
        nextc()
50 }
}

```

```

$1 == "b" {
    b++
55 if ($2 ~ /__epilog_.*/) {
        pop++
        nxt()
    } else {
        nextc()
    }
60 }

```

```

$1 == "beq" || $1 == "bne" {
    if ($2 !~ /__epilog_.*/) {
        beq++
    }
}

```

```

    nextc()
} else {
    next()
}
5 }
$1 == "bgt" || $1 == "ble" || $1 == "bge" || $1 == "blt" {
    if ($2 !~ /__epilog_.*/) {
        bgt++
        nextc()
10    } else {
        next()
    }
}
$1 == "bhi" || $1 == "bhs" || $1 == "blo" {
15    if ($2 !~ /__epilog_.*/) {
        bhi++
        next()
    } else {
        next()
20    }
}
$1 == "bpl" || $1 == "bmi" {
    if ($2 !~ /__epilog_.*/) {
        bpl++
25        next()
    } else {
        next()
    }
}
30 $1 == "jeq" || $1 == "jne" {
    if ($2 ~ "blink") {
        beq++
        nextc()
    }
35    next()
}
$1 == "jgt" || $1 == "jle" || $1 == "jge" || $1 == "jlt" {
    if ($2 ~ "blink") {
        bgt++
40        nextc()
    }
    next()
}
$1 == "j" {
45    if ($2 ~ "blink") {
        jblink++
        nextc()
    }
    if ($2 ~ reg) {
50        jr++
        nextc()
    }
    next()
}
55 $1 == "jl" {
    if ($2 ~ reg) {
        jlr++
        nextc()
    }
60    next()
}
$1 == "ld" {
    if ($2 ~ reg) {

```



```

    if ($3 == "[%sp,") {
        ldwsp++
        nxt()
    }
5   if ($3 == "[%gp,") {
        ldwgp++
        nextc()
    }
10  if ($3 ~ reg) {
        ldwr++
        if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 32)) {
            ldwr32++
            nextc()
        }
15  if ($4 ~ reg) {
            ldwabc++
            nxt()
        }
        next()
20  }
    }
    next()
}
$1 == "ldb" {
25  if ($2 ~ reg) {
        ldb++
        if ($3 == "[%fp,") {
            ldbfp++
            if (($4+0) >= -32 && ($4+0) <= -4) {
30          ldbfp32++
                next()
            }
        }
        next()
    }
35  if ($3 == "[%sp,") {
        ldbsp++
        next()
    }
    if ($3 == "[%gp,") {
40  ldbgp++
        next()
    }
    if ($3 ~ reg) {
        ldbr++
45  if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 16)) {
            ldbr16++
            nextc()
        }
        if ($4 ~ reg) {
50  ldbabc++
            next()
        }
        next()
    }
55  }
    next()
}
/st.%blink, "[%sp, 4\]/ {
    stblink++
60  nextc()
}
$1 == "st" {
    if ($2 ~ reg) {

```

```

    st++
    if ($3 == "[%fp,") {
#   stfpa[$4]++
    stfp++
5   if (($4+0) >= -32 && ($4+0) <= -4) {
        stfp32++
        nextc()
    }
    next()
10  }
    if ($3 == "[%sp,") {
#   stspa[$4]++
    stsp++
    next()
15  }
    if ($3 == "[%gp,") {
        stgp++
        next()
    }
20  if ($3 ~ reg) {
#   stra[$4]++
    str++
    if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 64)) {
25      str64++
        nextc()
    }
    next()
    }
30  }
    next()
}
$1 == "stw" {
    if ($2 ~ reg) {
        stw++
35      if ($3 == "[%fp,") {
#   stwfpa[$4]++
        stwfp++
        if (($4+0) >= -32 && ($4+0) <= -4) {
            stwfp32++
40          next()
        }
        next()
    }
    if ($3 == "[%sp,") {
45  #   stwspa[$4]++
        stwsp++
        next()
    }
    if ($3 == "[%gp,") {
50      stwgp++
        next()
    }
    if ($3 ~ reg) {
#   stwra[$4]++
55      stwr++
        if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 16)) {
            stwr16++
            nextc()
        }
60      next()
    }
}
next()

```

```

    }
    $1 == "stb" {
        if ($2 ~ reg) {
            stb++
5         if ($3 == "[%fp,") {
            #   stbfpa[$4]++
            stbfp++
            if (($4+0) >= -32 && ($4+0) <= -4) {
10             stbfp32++
            nxt()
            }
            nxt()
        }
        if ($3 == "[%sp,") {
15         #   stbspa[$4]++
            stbsp++
            nxt()
        }
        if ($3 == "[%gp,") {
20             stbgp++
            nxt()
        }
        if ($3 ~ reg) {
            #   stbra[$4]++
25             stbr++
            if ($3 ~ /\|/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 8)) {
                stbr8++
                nextc()
            }
30             nxt()
        }
        }
        nxt()
    }
35 $1 == "mov.f" {
    if ($2 == "0," && $3 ~ reg) {
        movf0r++
        nextc()
    } if ($2 == "0," && $3 ~ regh) {
40         movf0h++
        nextc()
    }
    nxt()
}
45 $1 == "mov" {
    if ($3 ~ /^-?[0-9]/) {
        movi++
        movia[$3]++
        if ($2 ~ reg) {
50             if ($3 >= 0 && $3 < 64) {
                movi64++
                nextc()
            }
        }
        if (pete) {
55             if ($2 ~ reg01 && $3 >= 0 && $3 < 128) {
                movi64p++
                nextc()
            }
        }
        if ($2 ~ reg23 && $3 >= 0 && $3 < 64) {
60             movi64p++
            nextc()
        }
        if ($2 ~ reg1316 && $3 >= 0 && $3 < 32) {

```



```

}
if ($4 ~ /^-?[0-9]/) {
  if ($2 ~ reg) {
    if ($3 ~ reg) {
      5      if ($4 >= -8 && $4 < 0) {
        subabi8++
        nextc()
      }
      10      if ($4 >= 1 && $4 <= 8) {
        addabi8++
        nextc()
      }
    }
    15    if ($3 ~ "%fp") {
      if ($4 >= -32 && $4 < 0) {
        addfpi32++
        nextc()
      }
    }
    20    if ($3 ~ /^%r([12][0-9])/ && $4 >= -512 && $4 < 512) {
      addrpc++
      nextc()
    }
  }
  25  next()
}
if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
  addrrrr++
  30  nextc()
}
$1 == "sub" {
  if ($4 ~ /^-?[0-9]/) {
    subi++
    35    if ($2 == $3) {
      # subia[$4]++
      if ($3 ~ reg) {
        if ($4 >= -32 && $4 < 0) {
          addi32++
          40          nextc()
        }
        if ($4 >= 0 && $4 < 32) {
          subi32++
          45          nextc()
        }
      }
    }
  }
  if ($2 ~ reg) {
    if ($3 ~ reg) {
      50      if ($4 >= -8 && $4 < 0) {
        addabi8++
        nextc()
      }
      if ($4 >= 1 && $4 < 8) {
        55      subabi8++
        nextc()
      }
    }
  }
  60  next()
}
if ($2 == $3 && $2 == ($4 ",")) {
  if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {

```

```

        subaaa++
        nextc()
    }
    if ($2 ~ regh && $3 ~ regh && $4 ~ regh) {
5      subhhh++
        nextc()
    }
}
10 if ($2 ~ reg) {
    subr++
    if ($2 == $3) {
        if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
            subaab++
            nextc()
15        }
        if ($2 ~ reg && $3 ~ reg && $4 ~ regh) {
            subrrh++
            nextc()
        }
20    if ($2 ~ reg && $3 ~ regh && $4 ~ reg) {
        subrrh++
        nextc()
    }
}
25 if ($3 ~ reg && $4 ~ reg) {
    subrrr++
    nextc()
}
30 nextc()
}
}
$1 == "sub.f" {
    if ($2 == "0,") {
        if ($3 ~ reg && $4 ~ reg) {
35          cmprr++
            nextc()
        }
        if ($4 ~ /^-?[0-9]/) {
            cmpi++
40          # cmpia[$4]++
            if ($3 ~ reg) {
                if ($4 >= 0 && $4 < 64) {
                    cmpi64++
                    nextc()
45                }
                if (pete) {
                    if ($3 ~ reg01 && $4 >= 0 && $4 < 128) {
                        cmpi64p++
                        nextc()
50                    }
                    if ($3 ~ reg23 && $4 >= 0 && $4 < 64) {
                        cmpi64p++
                        nextc()
                    }
55                if ($3 ~ reg1316 && $4 >= 0 && $4 < 32) {
                    cmpi64p++
                    nextc()
                }
            }
60        }
    }
    nextc()
}
if ($3 ~ reg) {

```

```

        if ($4 ~ regh) {
            cmprh++
            nextc()
        }
5      }
        if ($3 ~ regh) {
            if ($4 ~ reg) {
                cmphr++
                nextc()
            }
10     }
        }
        next()
    }
15  $1 == "sub.ne" {
        if ($2 == $3 && $2 == ($4 ",")) {
            if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
                subneaaa++
                nextc()
            }
20     }
        }
        next()
    }
25  $1 == "sub.eq" {
        if ($2 == $3 && $2 == ($4 ",")) {
            if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
                subeqaaa++
                nextc()
            }
30     }
        }
        next()
    }
35  $1 == "asl" {
        if ($4 ~ /^-?[0-9]/) {
            asli++
            if ($2 == $3) {
#         aslia[$4]++
                if ($3 ~ reg) {
                    if ($4 >= 1 && $4 <= 8) {
40                 asli8++
                    }
                    if ($4 >= 1 && $4 < 32) {
                        asli32++
                    }
45                 nextc()
            }
        }
        if ($2 ~ reg) {
            if ($3 ~ reg && $4 >= 2 && $4 < 3) {
50                 aslab2++
                nextc()
            }
        }
        next()
55  }
        if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
            aslaab++
            nextc()
        }
60  if ($2 ~ reg && $3 ~ reg && $4 !~ reg) {
            aslabl++
            nextc()
        }

```

```

}
$1 == "asr" {
  if ($4 ~ /^-?[0-9]/) {
    asri++
5    if ($2 == $3) {
#    asria[$4]++
      if ($3 ~ reg) {
        if ($4 >= 1 && $4 <= 8) {
10          asri8++
        }
        if ($4 >= 1 && $4 < 32) {
          asri32++
        }
        nextc()
15      }
    }
    if ($2 ~ reg) {
      if ($3 ~ reg && $4 >= 2 && $4 < 3) {
20        asrab2++
        nextc()
      }
    }
    next()
  }
25  if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
    asraab++
    nextc()
  }
  if ($2 ~ reg && $3 ~ reg && $4 !~ reg) {
30    asrab1++
    nextc()
  }
}
$1 == "lsr" {
35  if ($4 ~ /^-?[0-9]/) {
    lsri++
    if ($2 == $3) {
#    lsria[$4]++
      if ($3 ~ reg) {
40        if ($4 >= 1 && $4 <= 8) {
          lsri8++
        }
        if ($4 >= 1 && $4 < 32) {
          lsri32++
45        }
        nextc()
      }
    }
    if ($2 ~ reg) {
50      if ($3 ~ reg && $4 >= 2 && $4 < 3) {
        lsrab2++
        nextc()
      }
    }
    next()
55  }
  if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
    lsraab++
    nextc()
60  }
  if ($2 ~ reg && $3 ~ reg && $4 !~ reg) {
    lsrab1++
    nextc()
  }
}

```



```

    }
}
$1 == "mul64" {
    if ($2 == "0,") {
5        if ($4 ~ /^-?[0-9]/) {
            muli++
        #    mulia[$4]++
            if ($3 ~ reg) {
10                if ($4 >= 0 && $4 < 32) {
                    muli32++
                    nextc()
                }
            }
        }
15        if ($3 ~ reg && $4 ~ reg) {
            mul0ab++
            nextc()
        }
    }
20    next()
}
$1 == "and.f" {
    if ($2 == "0,") {
25        if ($4 ~ /^-?[0-9]/) {
            andfi++
        #    andfia[$4]++
            if ($3 ~ reg) {
                if ($4 >= 0 && $4 < 32) {
30                    andfi32++
                    nextc()
                }
            }
        }
35        if ($3 ~ reg && $4 ~ reg) {
            andfab++
            nextc()
        }
    }
40    next()
}
$1 == "and" {
    if ($2 == $3 || $2 == ($3 ",") || $2 == ($4 ",")) {
        if ($4 ~ /^-?[0-9]/) {
45            andi++
        #    andia[$4]++
            if ($3 ~ reg) {
                if ($4 >= 0 && $4 < 32) {
50                    andi32++
                    nextc()
                }
            }
        }
55        if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
            andaab++
            nextc()
        }
    }
    if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
60        andrrr++
        nextc()
    }
}
$1 == "extb" {

```

```

    if ($2 == ($3 ",")) {
        if ($2 ~ reg && $3 ~ reg) {
            extbr++
            nextc()
5        }
    }
    next()
}
$1 == "extw" {
10    if ($2 == ($3 ",")) {
        if ($2 ~ reg && $3 ~ reg) {
            extwr++
            nextc()
        }
15    }
    next()
}
$1 == "sexb" {
    if ($2 == ($3 ",")) {
20        if ($2 ~ reg && $3 ~ reg) {
            sexbr++
            nextc()
        }
    }
25    next()
}
$1 == "sexw" {
    if ($2 == ($3 ",")) {
30        if ($2 ~ reg && $3 ~ reg) {
            sexwr++
            nextc()
        }
    }
35    next()
}
($2 == $3 || $2 == ($3 ",") || $2 == ($4 ",")) {
    if ($1 == "add" || $1 == "sub" || $1 == "and" || $1 == "or" || $1 == "xor" ||
$1 == "asl" || $1 == "asr" || $1 == "lsr") {
40        if ($2 ~ reg) {
            if ($2 == $3) {
                if ($4 ~ reg) {
                    opaab[$1]++
                    nextc()
                }
            } else {
45                if ($3 ~ reg && $2 == ($4 ",")) {
                    opaab[$1]++
                    nextc()
                }
            }
50        }
    }
}
55 {
    next()
    # print $0
}
60 END {
    if (1) {
        OFS = "\t"
        # print "\nopaab"
    }
}

```

```

    for (i in opaab) {
        if (i == "add" || i == "sub" || i == "and" || i == "or" || i == "xor" || i ==
"asl" || i == "asr" || i == "lsr") {
            print i, opaab[i], int(opaab[i]*1000/NR)/10
5        }
    }
    # print "\nldfpa"
    # for (i in ldfpa) print i, ldfpa[i]
    # print "\nstfpa"
10    # for (i in stfpa) print i, stfpa[i]
    # print "\nldr0a"
    # for (i in ldr0a) print i, ldr0a[i]
    # print "\nmovia"
    # for (i in movia) print i, movia[i]
15    # print "\naddia"
    # for (i in addia) print i, addia[i]
    # print "\nsubia"
    # for (i in subia) print i, subia[i]
    # print "\ncmpia"
20    # for (i in cmpia) print i, cmpia[i]

    for (i in calls) {
        # print i, calls[i]
        if (calls[i] > 1) {
25            calls2 += (calls[i]-2)
        }
        callsall += calls[i]
    }
    # print "callsall", callsall, int(callsall*1000/NR)/10
30    # print "calls2", calls2, int(calls2*1000/NR)/10

    # bl = calls2
    bl = bl - push
    b = b - pop
35    print "bl", bl, int(bl*1000/NR)/10
    # print "push", push, int(push*1000/NR)/10

    print "b", b, int(b*1000/NR)/10
40    # print "pop", pop, int(pop*1000/NR)/10
    print "beq", beq, int(beq*1000/NR)/10
    print "bgt", bgt, int(bgt*1000/NR)/10
    print "bhi", bhi, int(bhi*1000/NR)/10
    print "bpl", bpl, int(bpl*1000/NR)/10

45    print "stblink", stblink, int(stblink*1000/NR)/10
    print "jblink", jblink, int(jblink*1000/NR)/10
    print "jlr", jlr, int(jlr*1000/NR)/10
    print "jlr", jlr, int(jlr*1000/NR)/10

50    print "movr", movr, int(movr*1000/NR)/10
    print "movf0r", movf0r, int(movf0r*1000/NR)/10
    print "movf0h", movf0h, int(movf0h*1000/NR)/10
    print "movrh", movrh, int(movrh*1000/NR)/10
55    print "movhr", movhr, int(movhr*1000/NR)/10

    print "cmprh", cmprh, int(cmprh*1000/NR)/10
    print "cmphr", cmphr, int(cmphr*1000/NR)/10
    print "cmpr", cmpr, int(cmpr*1000/NR)/10

60    print "cmpi64", cmpi64, int(cmpi64*1000/NR)/10
    print "cmpi64p", cmpi64p, int(cmpi64p*1000/NR)/10
    print "movi64", movi64, int(movi64*1000/NR)/10
    print "movi64p", movi64p, int(movi64p*1000/NR)/10

```

```

print "addi32", addi32, int(addi32*1000/NR)/10
print "subi32", subi32, int(subi32*1000/NR)/10
5  print "addabi8", addabi8, int(addabi8*1000/NR)/10
    print "subabi8", subabi8, int(subabi8*1000/NR)/10

    print "subneaaa", subneaaa, int(subneaaa*1000/NR)/10
10  print "subeqaaa", subeqaaa, int(subeqaaa*1000/NR)/10

    print "subhhh", subhhh, int(subhhh*1000/NR)/10
    print "subaaa", subaaa, int(subaaa*1000/NR)/10
    print "subaab", subaab, int(subaab*1000/NR)/10
    print "subrrr", subrrr, int(subrrr*1000/NR)/10
15  print "addaab", addaab, int(addaab *1000/NR)/10
    print "addrrr", addrrr, int(addrrr *1000/NR)/10
    print "addrhh", addrhh, int(addrhh *1000/NR)/10

    print "asli8", asli8, int(asli8*1000/NR)/10
20  # print "asli32", asli32, int(asli32*1000/NR)/10
    print "aslab1", aslab1, int(aslab1*1000/NR)/10
    print "aslab2", aslab2, int(aslab2*1000/NR)/10
    print "aslaab", aslaab, int(aslaab*1000/NR)/10

25  print "asri8", asri8, int(asri8*1000/NR)/10
    # print "asri32", asri32, int(asri32*1000/NR)/10
    print "asrab1", asrab1, int(asrab1*1000/NR)/10
    print "asrab2", asrab2, int(asrab2*1000/NR)/10
    print "asraab", asraab, int(asraab*1000/NR)/10

30  print "lsri8", lsri8, int(lsri8*1000/NR)/10
    # print "lsri32", lsri32, int(lsri32*1000/NR)/10
    print "lsrab1", lsrab1, int(lsrab1*1000/NR)/10
    print "lsrab2", lsrab2, int(lsrab2*1000/NR)/10
35  print "lsraab", lsraab, int(lsraab*1000/NR)/10

    print "andi32", andi32, int(andi32*1000/NR)/10
    print "andfi32", andfi32, int(andfi32*1000/NR)/10
    print "andaab", andaab, int(andaab *1000/NR)/10
40  print "andfab", andfab, int(andfab *1000/NR)/10

    print "mul0ab", mul0ab, int(mul0ab *1000/NR)/10
    print "muli32", muli32, int(muli32 *1000/NR)/10

45  print "ldabc", ldabc, int(ldabc *1000/NR)/10
    print "ldbabc", ldbabc, int(ldbabc *1000/NR)/10
    print "ldwabc", ldwabc, int(ldwabc *1000/NR)/10
    print "ldr64", ldr64, int(ldr64 *1000/NR)/10
    print "ldr64p", ldr64p, int(ldr64p *1000/NR)/10
50  print "ldwr32", ldwr32, int(ldwr32 *1000/NR)/10
    print "ldbr16", ldbr16, int(ldbr16 *1000/NR)/10
    print "str64", str64, int(str64 *1000/NR)/10
    print "stbr8", stbr8, int(stbr8 *1000/NR)/10
    print "stwr16", stwr16, int(stwr16 *1000/NR)/10
55  print "ldrpc", ldrpc, int(ldrpc *1000/NR)/10
    print "addrpc", addrpc, int(addrpc *1000/NR)/10

    print "ldfp32", ldfp32, int(ldfp32*1000/NR)/10
60  print "stfp32", stfp32, int(stfp32*1000/NR)/10
    print "addfpi32", addfpi32, int(addfpi32*1000/NR)/10

    print "ldgp", ldgp, int(ldgp*1000/NR)/10

```

```
print "stgp", stgp, int(stgp*1000/NR)/10
```

```
print "extbr", extbr, int(extbr*1000/NR)/10
```

```
print "extwr", extwr, int(extwr*1000/NR)/10
```

```
print "sexbr", sexbr, int(sexbr*1000/NR)/10
```

```
print "sexwr", sexwr, int(sexwr*1000/NR)/10
```

```
# print "movi", movi, "movi64", movi64, "movi128", movi128
```

```
# print "addi", addi, "addi32", addi32, "addi64", addi64, "addi128", addi128
```

```
# print "subi", subi, "subi32", subi32, "subi64", subi64, "subi128", subi128  
}
```

```
#function p(a, b) {
```

```
# print "a", b, int(b*100/NR)
```

```
#}
```

```
#!/(j|jl|b|bl)(ge|gt|le|lt|ne|eq|pl|mi|hi|hs|lo|ls)?\.d/ {
```

```
# stored = $0
```

```
# sub(/\.d/, "", stored)
```

```
# getline
```

```
# print $0
```

```
# print stored
```

```
# nextc()
```

```
#}
```

```
#
```

```
{ print $0 }
```

APPENDIX IV

Copyright © 2000-2001 ARC International plc. All rights reserved.

```

#
# Confidential Information
# Limited Distribution to Authorized Persons Only
# Created 2000 and Protected as an Unpublished Work
5 # Under the U.S. Copyright act of 1976.
# Copyright © 2000-2001 ARC CORES LTD
# All Rights Reserved
#
# SCCS release : %M% %I% %G%
10 #
# Description : Script to mark pairs of compress instructions
# for an ARC assembler file.
#
#
15 #
#--
=====
=====--#

20 /^c/ {
    a=$0
    nra=NR
    getline b
    if (b ~ /^c/) {
25     c++
    print "p" a
    print "p" b
    next
    } else {
30     print a
    print b
    next
    }
    }
35 {
    print $0
    }

40 END {
    # print c
    }

```

APPENDIX V

Copyright © 2000-2001 ARC International plc. All rights reserved.

APPENDIX VI

Copyright © 2000-2001 ARC International plc. All rights reserved.

CONFIDENTIAL

```
rem
rem
rem          Confidential Information
rem          Limited Distribution to Authorized Persons Only
rem          Created 2000 and Protected as an Unpublished Work
5 rem          Under the U.S.Copyright act of 1976.
rem          Copyright © 2000-2001 ARC CORES LTD
rem          All Rights Reserved
rem
rem
rem SCCS release : %M% %I% %G%
10 rem
rem Description : Script to generate a report on analysis of
rem               an ARC assembler file
rem
rem
rem
15 rem
rem--
=====
=====--#

20 @echo off
del /q *.r
for %x in (@%) do echo %x >%x.r
for %x in (@%) do awk -f \awk\REPORT.AWK %x\f >>%x.r
for %x in (@%) do (pushd ^ cd %x ^ awk -f \awk\ratio.AWK %x\cp
25 >>..\%x.r ^ popd)
rem awk '{printf "\t%5s" $1}' %1
cat nl isa14 >i
pr -m -t i *.r
□
30
```

APPENDIX VII

Copyright © 2000-2001 ARC International plc. All rights reserved.

=====

```
rem
rem
rem          Confidential Information
rem          Limited Distribution to Authorized Persons Only
rem          Created 2000 and Protected as an Unpublished Work
5 rem          Under the U.S.Copyright act of 1976.
rem          Copyright © 2000-2001 ARC CORES LTD
rem          All Rights Reserved
rem
rem SCCS release : %M% %I% %G%
10 rem
rem Description  : Script to generate a report on analysis of
rem               an ARC assembler file
rem
rem
15 rem
rem--
=====
=====--#
20 call rep apps1 >t1
call rep apps2 >t2
pr -w 160 -s -m -t t1 t2 >t
expand -t 8 t >tt
□
25
```

APPENDIX VIII

Copyright © 2000-2001 ARC International plc. All rights reserved.

```

#
#
#           Confidential Information
#           Limited Distribution to Authorized Persons Only
#           Created 2000 and Protected as an Unpublished Work
5  #           Under the U.S. Copyright act of 1976.
#           Copyright© 2000-2001 ARC CORES LTD
#           All Rights Reserved
#
# SCCS release : %M% %I% %G%
10 #
# Description : Script to print a report for usage of
#              specified ARC instruction formats from an ISA file
#              and an ARC assembler file.
#
15 #
#
#--
=====
=====--#
20 BEGIN {
    isa = "isa14"
    while (getline <isa) { format[$1]=1 }
    OFS="\t"
25 }

{
    for (i in format) {
        if ($1 == i) {
30             t += $3
            if ($3 == "") {print "0"} else {print $3}
            # if ($3 == "") {print $1,"0"} else {print $1,$3}
        }
    }
35 }

END {
    print t
40 # for (i in format) { print i}
}
□

```

APPENDIX IX

Copyright © 2000-2001 ARC International plc. All rights reserved.


```

rem
rem                               Confidential Information
rem               Limited Distribution to Authorized Persons Only
rem       Created 2000 and Protected as an Unpublished Work
5 rem               Under the U.S.Copyright act of 1976.
rem               Copyright © 2000-2001 ARC CORES LTD
rem               All Rights Reserved
rem

```

```

10 rem SCCS release : %M% %I% %G%
rem
rem Description : Script to strip out non-instruction lines
rem               from an ARC assembler file
rem
rem

```

```

15 rem
rem--

```

```

=====
=====--#

```

```

20 @echo off
egrep -v "^( $|;|.|\\.|\\.|[~a-zA-Z]|.|[0-9])" d >dd
wc -l dd
grep -c "^\" dd
25 grep -c "<= Compressable" dd

```

```

□

```

APPENDIX X

Copyright © 2000-2001 ARC International plc. All rights reserved.

```

rem
rem          Confidential Information
rem          Limited Distribution to Authorized Persons Only
rem          Created 2000 and Protected as an Unpublished Work
5 rem          Under the U.S.Copyright act of 1976.
rem          Copyright © 2000-2001 ARC CORES LTD
rem          All Rights Reserved
rem

```

```

rem SCCS release : %M% %I% %G%

```

```

10 rem
rem Description : Script to strip out more non-instruction lines
rem              from an ARC assembler file
rem
rem

```

```

15 rem
rem--

```

```

=====
=====--#

```

```

20 sed "s/^\[///" dd |sed "s/ \] $//" |grep -v "^.;" |tr "|" "\n" |sed "s/^\[/"
    |sed "s/ ;.*//" |awk -f delay.awk >dds

```

APPENDIX XI

Copyright © 2000-2001 ARC International plc. All rights reserved.

T
h
e
C
o
n
f
i
d
e
n
t
i
a
l
I
n
f
o
r
m
a
t
i
o
n

```
#
#
#               Confidential Information
#       Limited Distribution to Authorized Persons Only
#       Created 2000 and Protected as an Unpublished Work
5  #       Under the U.S. Copyright act of 1976.
#       Copyright © 2000-2001 ARC CORES LTD
#       All Rights Reserved
#
# SCCS release : %M% %I% %G%
10 #
# Description   : Script to place instructions that are in a delay
#                 slot to before the branch and remove the ".d"
#                 from the branch of an ARC assembler file
#
15 #
#
#--
=====
=====--#
20 /(j|j1|b|b1)(ge|gt|le|lt|ne|eq|pl|mi|hi|hs|lo|ls|pnz)?\.d/ {
    stored = $0
    sub(/\.d/, "", stored)
    getline
25    print $0
    print stored
    next
}
30 { print $0 }

35

40
```

APPENDIX XII

Copyright © 2000-2001 ARC International plc. All rights reserved.

Instruction formats

i = instruction opcode
a = register (r0-3,r13-16)
b = register (r0-3,r13-16)
5 c = register (r0-3,r13-16)
h = register high (r0-r31)
q = condition code
u = unsigned immediate
10 s = signed immediate

Op	Format	Instruction	Operands	Comment
0	iiiiqqssssssss	bal/beq/bne	s8	; if cc pc=(pc&0xffffffffc)+(s8<<1)
	iiiii1qqssssss	bgt/bge/blt/ble	s6	; if cc pc=(pc&0xffffffffc)+(s6<<1)
1	iiisssssssssss	bl	s10	; blink=pc; pc=(pc&0xffffffffc)+(s10<<2)
15	2	iiiiaaabbbiiiii op	a,a,b	; op = sub/and/or/xor/asl/asr/lshr/
			a,b,1	; asl1/asr1/
			a,b,2	; asl2/asr2/
			0,a,b	; and.f/mul64/?/?/s_op
20		iiiiaaaii1111 s_op	a,a	; s_op=extb/extw/sexb/sexw/
			[a]	; j/jl/
			a,a,a	; sub.ne/i_op
		iiiiiii1111111 i_op3		; i_op=brk/j [blink]/st blink[sp,4]/
	3	iiiiaaaiuuuuuu mov/cmp	a,u6	
	4	iiiiaaaiuuuuuu add/sub/?/?	a,a,u5	
25	5	iiiiaaahhh00hh mov	a,h	
		iiiiaaahhh01hh add	a,a,h	
		iiiiaaahhh11hh mov/cmp	h,a	
	6	iiiiaaa000iuuu ld/st	a,[fp, -u3]	; a=mem[fp - (u3 << 2)]
		iiiiaaa001iuuu add/?	a,[fp, -u3]	; a=fp - (u3 << 2)
30		iiiiaaaiuuuuuu asl/asr/lshr	a,a,u5	
	7	iiiiaaabbbuuuu ld	a,[b,u4]	; a=mem[b + (u4<<2)]
	8	iiiiaaabbbuuuu ldb	a,[b,u4]	; a=mem[b + u4]
	9	iiiiaaabbbuuuu ldw	a,[b,u4]	; a=mem[b + (u4<<1)]
	A	iiiiaaabbbuuuu st	a,[b,u4]	; a=mem[b + (u4<<2)]
35	B	iiiiaaabbb0uuu stb	a,[b,u3]	; a=mem[b + u3]
		iiiiaaabbb1uuu stw	a,[b,u3]	; a=mem[b + (u3<<1)]
	C	iiiiaaabbbiuuu add/sub	a,b,u3	; a=b op u3
	D	iiiiaaaauuuuuuu ld	a,[pc,u7]	; a=mem[(pc&0xffffffffc)+(u7 << 2)]
	E	iiiiaaaauuuuuuu ld	a,[gp,u7]	; a=mem[gp + (u7 << 2)]
40	F	iiiixxxxxxxxxx reserved		

Other possible formats:

iiiiaaabbb0ccc ld a,[b, c] ; a=mem[b + c]
iiiiaaabbb1ccc add a,b,c ; a=b+c
45 iiiiaaaauuuuuuu add a,pc,u7 ; a=(pc&0xffffffffc)+(u7 << 2)